Evaluating State Machine Replication Over a WAN

João Sousa and Alysson Bessani
University of Lisbon, Faculty of Sciences, LaSIGE
Lisbon, Portugal

***Abstract*—This paper presents preliminary results regarding the performance of a Byzantine fault-tolerant (BFT) state machine replication (SMR) protocol executing over a wide area network (WAN). Some well known optimizations were implemented and evaluated in the protocol. Our preliminary results suggest SMR can show stable performance across WANs, but some of the optimizations evaluated may be ineffective in practice.**

I. Introduction

The state machine replication technique [17], [28] enables an arbitrary number of client to issue requests into a set of replicas. These replicas implement a stateful service that updates its state after receiving those requests. The goal of this technique is to enforce strong consistency within the service, by making it completely and accurately replicated at each replica. The key to make the state evolve with such consistency is to execute a distributed protocol that forces each operation sent by clients to be delivered at each replica in the exact same order. When clients get the service's reply, their requests are already reflected in the service's state.

This paper reports preliminary results regarding the performance of a Byzantine fault-tolerant (BFT) state machine replication (SMR) protocol executing over a wide area network (WAN). Besides evaluating the protocol's performance, we also implemented and evaluated some optimizations from the literature which aim to render executions of SMR protocols more stable and efficient across WAN environments. The experiments presented in this paper were made using BFT-SMaRt [1], a library that implements a generic BFT-SMR protocol similar to the well known PBFT [10].

The main motivation behind these preliminary experiments is twofold. First, we wanted to evaluate the behavior of a generic BFT-SMR protocol when deployed on a large scale environment. Secondly, there is significant research in optimizing SMR for large scale environments (e.g., [4], [21], [27]) which propose several optimizations to SMR aimed at improving performance in a WAN. We decided to perform our own evaluation of some of the optimizations introduced in those works. We did this by implementing them in a generic BFT-SMR protocol (used by BFT-SMaRt).

Both LAN and WAN settings assume the same system model; some hosts are assumed to be malicious, and the network connecting each host does not offer any time guarantees. This is in accordance to the assumptions made by BFT-SMaRt's protocol [29]. The practical difference is that, in a WAN setting, network delay is much higher and variable, and message loss is much more frequent than it is in a LAN.

Our preliminary results suggest that a generic BFT-SMR protocols can display stable performance across WANs. However, out of the three optimizations evaluated, only one seemed to significantly improve the protocol's performance; the remaining two did not introduce any observable improvement. Further experiments will be required in order to reach conclusive results.

The rest of this paper is organized as follows. We discuss some related work in Section II. Section III presents the hypotheses we aim to investigate. Section IV describes the methodology used to conducted the experiments. We report our results for each hypothesis in Sections V, VI, VII and VIII. Finally, we discuss future work in Section IX and present our conclusions in Section X.

II. Related Work

In this section we give a brief overview of related work regarding Byzantine fault tolerance, state machine replication, wide area replication and wide area measurements.

a) Byzantine fault tolerance: Byzantine fault tolerance (BFT) is a sub-field of fault tolerance research within distributed systems. In classical fault tolerance, processes are assumed to fail only by stopping to execute. On the other hand, in the *Byzantine faults model*, processes of a distributed system are allowed to fail in a arbitrary way, i.e., a fault is characterized as any deviation from the specified algorithm/protocol imposed on a process. Thus, Byzantine fault tolerance is the body of techniques that aims at devising protocols, algorithms and services that are able to cope with such arbitrary behavior of the processes that comprise the system [20]. In Byzantine fault tolerance it is common practice to make assumptions as weak as possible, not only from the processes that comprise the system, but also from the network that connects them. In BFT, it is typically adopted either the partially synchronous or asynchronous system model [14].

b) State machine replication: The state machine replication (SMR) model was first proposed by Lamport in [17], and later generalized by Schneider in [28]. In this model, an arbitrary number of client processes send commands to a set of servers, which function as replicas of a stateful service that updates its state after processing commands sent by clients. The goal of this technique is to make the state at each replica to evolve in a consistent way, resulting in a service which is accurately replicated across all replicas. Since the service state was already updated by the time clients receive a response from the service, this technique is able to offer *strong consistency*. In order to enforce this behavior, it is necessary to satisfy the following properties: (1) If any two correct replicas r and r' apply operation o to state s, both r and r' will obtain state s'; (2) Any two correct replicas r and r' start with state s_0; (3) Any two correct replicas r and r' execute the same sequence of operations $O_0, ..., O_i$; (4) Operations from correct clients get executed.

Castro and Liskov showed that executing SMR under Byzantine faults is feasible, by presenting the PBFT protocol [10]. PBFT is capable of withstanding up to f Byzantine replicas out of at least $3f + 1$. The main difference between PBFT and previous proposals for BFT were that PBFT avoided expensive cryptographic operations such as digital signatures by using MAC vectors instead. PBFT spawned a *renaissance* in BFT research, and is considered the baseline for all BFT-SMR protocols published afterwards. Moreover, the idea of porting SMR protocols into WANs also found new momentum.

c) Wide Area Replication: Mencius [21] is a SMR protocol derived from Paxos [18], [19] which is optimized to execute in WANs. It can survive up to f crashed replicas out of at least $2f+1$. According to the paper, its rotating coordinator mechanism can significantly reduce clients' latency in WANs. Replicas take turns as the leader and propose client requests in their turns. Clients send requests to the replicas in their sites, which are submitted for ordering when the replicas becomes the leader.

Veronese et. al. introduced *Efficient Byzantine Agreement for Wide-Area Networks* (EBAWA) [27], a BFT-SMR protocol optimized for WANs. Since it uses the trusted/trustworthy USIG service introduced in [30], it requires only $2f + 1$ replicas to tolerate f Byzantine faults. Similarly to Mencius, it uses a rotating leader scheme to prevent a faulty leader from degrading system performance.

d) Measurements: The aforementioned replication protocols rely on some form of quorum systems to be capable of guaranteeing their safety properties. Given a set of hosts, a *quorum system* is a collection of sets of hosts (called *quorums*) such that any two quorums intersect in at least one common host [12], [13]. Since quorum systems are building blocks used to implement a variety of services (e.g., consensus [8], mutual exclusion [3], distributed access control [23]), there is interest in predicting their availability and performance in WANs.

The traditional approaches for evaluating quorum systems used to be either by analysis and/or by emulation. Amir et. al. proposed in [5] an empirical approach, which consisted of gathering uptime data from a real system consisting of multiple hosts. These hosts were scattered across two distinct sites which communicated with each other over the internet. The aforementioned data was obtained using a group communication framework responsible for generating uptime logs for each host in the experiment. These logs were then integrated into one single global log, which represented the availability of all hosts within some time period. According to the authors, the results obtained suggest that machine crashes are correlated, network partitions are frequent, and a system crash is a rare, yet possible event. They also confirm that, if network partitions are frequent, dynamic quorum systems (e.g., [15]) yield better availability than static quorums.

Bakr et. al. also employed empirical measurements in [6]. Whereas Amir et. al. investigated the *availability* of a quorum system, Bakr et. al. investigate the *latency* of distributed algorithms over the internet. Both works used real hosts to obtain their measurements, but the methodology was different: instead of using a communication group framework to construct logs, Bakr et. al. implemented their own daemons that would periodically initiate the algorithms, and keep track of the running time for each iteration. Furthermore, the hosts were geographically distributed across more than two sites. Upon evaluating some of these algorithms, the authors observed the message loss rate over the internet was not negligible. Moreover, algorithms with high message complexity display higher loss rate. This is evident in protocols that employ *all-to-all* communication, like PBFT and BFT-SMaRt.

However, the experiments performed in [6] did not assume quorum systems, i.e., each interaction of a communication round was only considered finished once every host received/replied to all messages. In a quorum system, each hosts only waits for a majority of hosts to receive/reply to all messages. The authors conducted further research in [7] considering quorum systems and how the number of hosts probed by a client impacts latency and availability. Furthermore, two quorum types were studied: (1) majority quorum systems, where the quorums are sets that include a majority of the hosts, and (2) crumbling walls quorum systems[1] [25], which uses smaller quorums with varying sizes. The authors argue that their results suggest that majority quorums do not perform well with small probe sets. Moreover, the authors also claim that increasing the size of the probe by as few as a single host can reduce latency by a considerable margin. They also argue that their results show that crumbling walls can display better latency than majority quorums, and also comparable availability. On the other hand, this study only investigated availability of quorums as a function of the probing performed by clients; the behavior of complex distributed protocols which execute over quorum systems was not explored.

Finally, it is worth mentioning the experiments presented in [5]–[7] were conducted from 1996 to 2008. Given the current advances in network infrastructures, the same experiments may yield different results if they were executed in 2013.

III. Hypotheses

In the experiments reported in this paper, we evaluated some protocol optimizations known in the literature, which are implemented by the SMR protocols described in Section II-c, and one optimization related to quorum systems proposed in [13], [26]. More precisely, we want to test the following hypotheses:

1) The leader location influences the observed latency of the protocol (Section V);
2) A bigger quorum size can reduce the observed latency (Section VI);
3) Read-only and tentative executions significantly reduces the observed latency (Section VII).

Finally, we evaluated the stability of BFT-SMaRt's protocol within a WAN, i.e., how much the latency observed by BFT-SMaRt clients vary across a long execution. More specifically, the following hypothesis was tested in Section VIII: *A generic BFT-SMR protocol is stable and predictable enough in a WAN environment and can be used to implement services capable of exhibiting satisfactory performance and usability*.

[1] In a crumbling wall, hosts are logically arranged in rows of varying widths, and a quorum is the union of one full row and a representative from every row below the full row.

IV. Methodology

The following experiments were conducted over the course of approximately 40 days on Planetlab [11], a distributed testbed scattered throughout the world, dedicated to computer networking and distributed systems research. A total of eight host were selected for these experiments. These hosts are listed in Table I, and were divided into groups A and B, each one executing a single instance of a simple benchmarking application implemented over BFT-SMaRt.

Group A is comprised of a set of 6 replicas, whereas group B is comprised of 4. Furthermore, replica 1 is represented in 2 hosts. The reasons for this are the following: (1) one extra host was necessary for the experiment described in Section VI, and (2) the original host for replica 1 (Italy, Parma) became unavailable during the course of the experiments. Hence, we needed to deploy a new host to take its place (Braga, Portugal).

No additional hosts were used to run the clients. Each host ran two processes simultaneously: one executed a BFT-SMaRt replica, and the other ran multiple threads which implemented BFT-SMaRt clients. Each client sent a request to the replicas every 2 seconds. A distinguished client at each host was programmed to write its observed latency into a log file. Each client issued a 4 kB request, and received a 4 kB reply.

Group A

Replica	Location	Hostname
0 (leader)	Birmingham, England	planetlab1.aston.ac.uk
1	Italy, Parma Portugal, Braga	planet1.unipr.it planetlab-um00.di.uminho.pt
2	Germany, Darmstadt	host2.planetlab.informatik.tu-darmstadt.de
3	Norway, Oslo	planetlab1.ifi.uio.no
4	Belgium, Namur	orval.infonet.fundp.ac.be

Group B

Replica	Location	Hostname
0 (leader)	Poland, Gliwice	plab4.ple.silweb.pl
1	Spain, Madrid	utet.ii.uam.es
2	France, Paris	ple3.ipv6.lip6.fr
3	Switzerland, Basel	planetlab-1.cs.unibas.ch

TABLE I. Hosts used in experiments

V. Leader Location

The goal of this first experiment is to observe how much the leader's proximity to an aggregate of clients can improve their observed latency. This is motivated by the fact that Mencius [21] and EBAWA [27] both use a rotating leader scheme to improve client's latency: if the leader is close to the clients, their messages arrive sooner and thus are ordered faster.

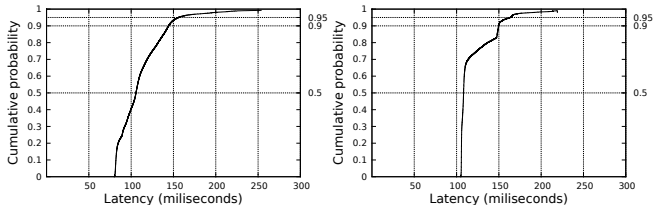
This experiment ran on Group B with an aggregate of 10 clients. The experiment was repeated 4 times, each one placing the aggregate in a distinct host. Each iteration took approximately one day to run. Only one client was launched on the rest of the hosts that did not had the aggregate.

Table II presents the average latencies observed by client in each distinct location. Each row depicts the observed latency for the client location, whereas each column depicts the location of the aggregate. In the case of Gliwice - where clients run in the same machine as the leader - the average latency was lowest when the aggregate was placed in the same host as the leader. Moreover, for the rest of the locations, latency was highest when they also hosted the aggregate (only exception being Basel). This indicates that moving an aggregate of clients close to the leader may improve latency. However, if the leader is in a distinct site, client latency tends to increase at the site that hosts the aggregate.

Client \ Aggregate	Gliwice	Madrid	Paris	Basel
Gliwice	112 ± 54.07	117 ± 211.02	113 ± 34.44	118 ± 39.61
Madrid	120 ± 44.78	122 ± 158.38	122 ± 150.28	90 ± 42.12
Paris	120 ± 68.64	123 ± 230.96	125 ± 90.71	94 ± 56.56
Basel	119 ± 182.55	122 ± 151.83	119 ± 33.50	96 ± 173.31

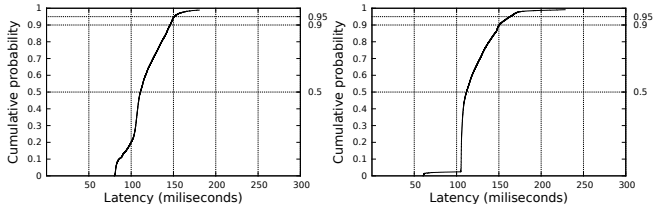
TABLE II. Average latency and standard deviation observed in Group B's distinguished clients, with the leader in Gliwice. All values are given in milliseconds.

Figures 1 and 2 illustrate the cumulative distribution for the latencies observed by distinguished clients in Gliwice and Madrid. Figure 1 illustrate the latency when the aggregate is close to the leader (both located in Gliwice), whereas 2 illustrate the latency observed once the leader and aggregate are detached (leader located in Gliwice, aggregate in Madrid).



(a) Gliwice. (b) Madrid.

Fig. 1. Cumulative distribution of latencies when aggregate is hosted in Gliwice (close to the leader).



(a) Gliwice. (b) Madrid.

Fig. 2. Cumulative distribution of latencies when aggregate is hosted in Madrid (far from the leader).

Despite placing the aggregate close to the leader, the above results suggest that doing so just barely improves client latency (e.g, moving the aggregate from Madrid to Gliwice results in a average improvement of less than 3%). There are two possible explanations for such small benefit: a) BFT-SMaRt's protocol needs to execute two *all-to-all* communication steps, which combined take much more time to finish than the time it takes for the clients to contact all replicas (regardless of their geographic location). Mencius and EBAWA are able to avoid such communication complexity because, unlike BFT-SMaRt, they do not assume a full BFT model (Mencius assumes crash faults and EBAWA uses a trusted component); b) The size of the message sent on the one-to-all communication step initialized by the leader is much larger than any of the all-to-all communication steps sent afterwards (since they only contain a cryptographic hash of the requests being ordered). The results shown in Section VII suggests this is the correct explanation. Given these results, it does not seem to be advantageous to periodically change the leader's location.

VI. Quorum Size

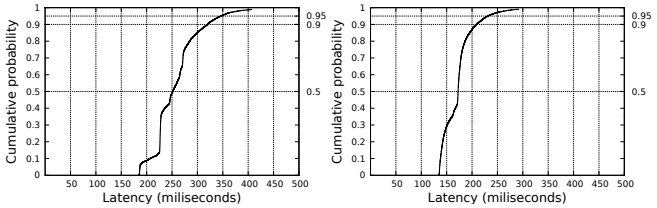
The purpose of this experiment was to observe how client latency is affected by the quorum size demanded by the protocol. This is motivated by the works of Gifford [13] and Pâris [26], which use voting schemes with additional hosts to improve availability. While Gifford makes all hosts hold a copy of the state with distinct voting weights, Pâris makes a distinction between hosts that hold a copy of the state and hosts that do not hold such copy, but still participate in the voting process (thus acting solely as witnesses).

This experiment ran on Group A, with replica 1 hosted in Braga. It was repeated twice: one using 4 hosts and other using 5. BFT-SMaRt was modified to use only 3 replicas out of the whole set in each iteration (thus, in each execution waiting for 3/4 and 3/5 replicas respectively). Each experiment executed for approximately 24 hours. Five clients were launched at each host, creating a total of 20 clients in the experiment.

Client Location	Quorum size	
	3/4	3/5
Birmingham	551 ± 1440.44	812 ± 1319.025
Braga	258 ± 132.83	171 ± 38.3
Darmstadt	266 ± 149.89	200 ± 291.88
Oslo	260 ± 131.027	203 ± 203.39

TABLE III. Average latency and standard deviation observed in Group A. All values are given in milliseconds.

The results shown in Table III display the average latency and standard deviation observed in the distinguished clients in both iterations. After running the experiment with one more replica in the group, both average latency and standard deviation decreased in all distinguished clients (with the exception of Birmingham). Since BFT-SMaRt still waits for the same number of replies in each communication step, the slowest replica within that set is replaced by the additional one, thus decreasing latency. Birmingham did not display this behavior because the host was visibly unstable throughout both iterations.



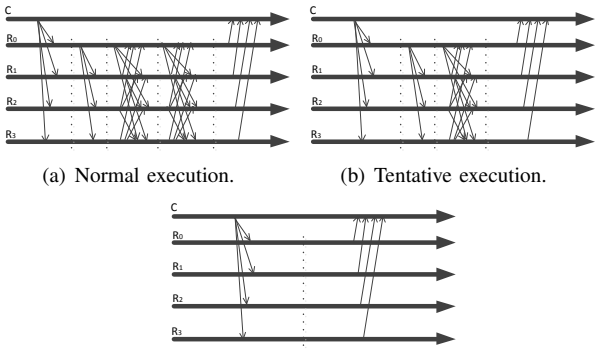
(a) Quorum size: 3/4. (b) Quorum size: 3/5.

Fig. 3. Cumulative distribution of latencies observed in Braga (group A).

The difference between both iterations is better illustrated in Figure 3, which displays the cumulative distribution function of the latency observed in Braga's distinguished client. When using 3/4 quorums, the 95th percentile of all latencies is approximately 350 milliseconds (Figure 3(a)), whereas when using 3/5 the same percentile is approximately 240 milliseconds (Figure 3(b)). This observation further indicates that using additional replicas can improve the latency observed by clients.

VII. Communication Steps

The purpose of the following experiments is to observe how client latency is affected by the amount of communication steps performed by the BFT-SMR protocol. More precisely, we wanted to observe how efficient *read-only* and *tentative* executions are in a WAN. These two optimizations are proposed in PBFT [10] to reduce latency. The message pattern for each of these optimizations is illustrated in Figure 4.



(a) Normal execution. (b) Tentative execution.

(c) Read-only execution.

Fig. 4. Message patterns evaluated.

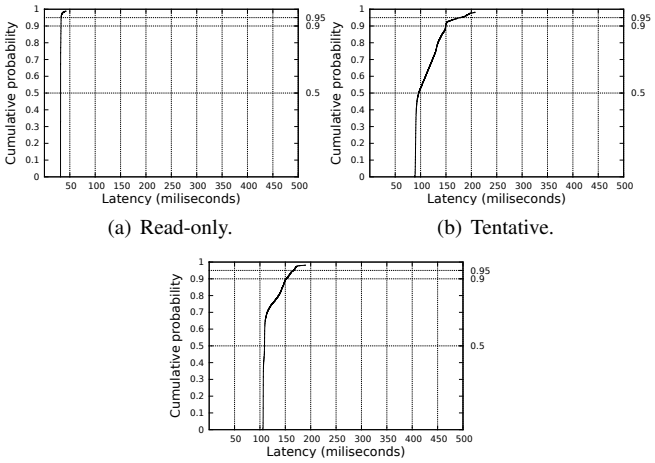
Figure 4(a) depicts the normal execution of our generic SMR protocol, which is comprised of 5 communications steps (as in BFT-SMaRt). Figure 4(b) displays the message pattern for tentative executions. This optimization reduces the number of communication steps from 4 to 5 by bypassing one of the all-to-all communication steps of normal execution. This optimization comes at the cost of potentially needing to perform a rollback on the application state. Figure 4(c) shows the message pattern for read-only executions. This optimization enables the clients to fetch a response from the service in only 2 communication steps (the client's request and the replicas' replies). However, this optimization can only be used to read the state from the service, and never to modify it.

The experiment here reported ran on Group B. To perform the tentative executions, BFT-SMaRt was modified to skip one of the all-to-all communication steps specified by its protocol [29]. Read-only executions were already implemented in BFT-SMaRt. Each iteration was executed for approximately 12 hours. Five clients were launched at each host, thus creating a total of 20 clients in the experiment.

Client Location	Execution Type		
	Read Only	Tentative	Normal
Gliwice	59 ± 11.14	100 ± 35.79	112 ± 29.33
Madrid	31 ± 6.86	112 ± 37.09	122 ± 183.54
Paris	25 ± 154.32	110 ± 39.38	118 ± 43.61
Basel	33 ± 224.7	110 ± 35.78	117 ± 30.315

TABLE IV. Average latency and standard deviation observed in Group B. All values are given in milliseconds.

Table IV shows the average latency and standard deviation observed by the distinguished clients in each iteration. Figure 5 depicts the cumulative distribution for latencies observed by Madrid's distinguished clients for each type of execution. Both Table IV and Figure 5 show that read-only execution significantly exhibits the lowest latency, finishing each execution faster than any of the other iterations (in Madrid's case, as less as 25.4% of the latency of normal execution).



(a) Read-only. (b) Tentative.

(c) Normal.

Fig. 5. Cumulative distribution of latencies observed in Madrid (group B).

Tentative execution also manages to reach lower latency values than normal execution does. However, even though this optimization omits an entire all-to-all communication step, the difference is not significant (less than 8% improvement on average). This may be explained by the fact that the size of the message sent on the one-to-all communication step initialized by the leader is much larger than any of the all-to-all communication steps performed afterwards (which only contains a cryptographic hash of the requests being ordered). In any case, the results for this particular experiment are inconclusive, and will be further investigated in future work.

VIII. BFT-SMaRt Stability

The goal of this experiment was to observe how stable BFT-SMaRt's protocol is once deployed in a WAN, and to find if there is a time interval within which it is highly likely to finish ordering requests. This was done by observing the latencies experienced by distinguished clients. None of the optimizations evaluated in Sections V-VII were used in this experiment.

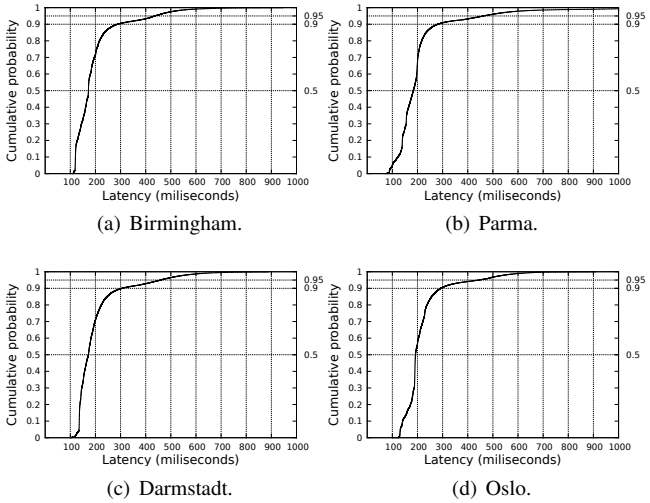
This experiment was executed within group A over a period of approximately 600 hours. In this experiment, replica 1 was hosted in Parma, Italy. Five clients were launched at each host, thus making a total of 20 clients in the experiment.

Client Location	Invocations	Average Latency	Median	90th	95th
Birmingham	717782	195 ± 144.61	172	287	436
Parma	918293	210 ± 237.81	183	280	462
Darmstadt	444054	203 ± 207.54	172	305	459
Oslo	708811	220 ± 166.16	192	293	444

TABLE V. Average latency, standard deviation, Median, 90th and 95th percentile observed in Group A. Values are given in milliseconds.

Table V shows the results for the average latency, standard deviation, Median, 90th and 95th percentile calculated at each distinguished client. It also discriminates the number of invocations performed by each one of them. Figure 6 plots the cumulative distribution for those latencies. During this 600-hour experiment, the average latency ranged from 195 to 220 milliseconds across all sites. Even though these averages fall in an acceptable range, their associated standard deviations are high, ranging from 144.51 to 237.81. This demonstrates that latency was quite variable during this experiment.

On the other hand, about 95% of the observed latencies fall under 462 milliseconds at all locations. This means that latency, albeit variable, rarely exceeds 500 milliseconds. Even though it is approximately 5 times higher than the ideal latency of 100 miliseconds identified in [9], [22], [24], clients received their response under much less than one second in the 95th percentile (which according to the same studies, is still sufficient for user satisfaction). This suggests that BFT-SMR protocols can be stable enough to be used across WANs and are able to reply to clients within a predictable time interval.



(a) Birmingham. (b) Parma.

(c) Darmstadt. (d) Oslo.

Fig. 6. Cumulative distribution of latencies of group A over the course of two weeks.

IX. Discussion & Future Work

Out of the three optimizations evaluated, only the quorum size was shown to be effective in improving the clients's latency. Our 600-hour experiment indicated that clients experience latency that remains within user satisfaction limits. Both optimizations tested in Sections V and VII seem to be ineffective due to the message size sent by clients. There is one more explanation for such lack of performance gain: a 4 kB payload is likely to cause packet fragmentation, which implies more probability of packet loss. Since we use TCP connections, the packets are retransmitted, but the remaining ones may get hold in TCP buffers. Given that we are working within a WAN, this is a possible explanation for these results. In the future we plan to repeat these experiments using 1 kB of payload in client messages.

However, these initial results only reflect the performance of a generic BFT SMR protocol within PlanetLab; they are dependent on both BFT-SMaRt's implementation and the testbed environment. To obtain conclusive results, it would be necessary to run these experiments in more testbeds such as EmuLab [2] or even across distinct cloud providers.

Nonetheless, we intend to conduct further investigation regarding all optimizations evaluated in this work. We wish to run all experiments during a larger time interval, using hosts scattered across all continents, instead of limiting the

scope of the experiments to Europe. We will also fiddle with more variables in each experiment. For instance, in Section V we only moved the aggregate's position; in a new iteration of the experiment, we will also want to observe how the leader position affects the results. We will evaluate quorum size with larger and smaller quorums, and evaluate the protocol's performance using speculative execution [16]. We will also evaluated more optimizations, such as sending client requests to only one replica instead of sending those requests to the whole set.

X. Conclusion

In this paper we have reported preliminary results from experiments conducted in PlanetLab executing a state machine replication protocol capable of withstanding Byzantine faults. These experiments were meant to find how would a standard BFT SMR protocol benefit from optimizations taken from the literature, and to learn how it would perform over a WAN environment. Albeit further work is necessary, these results indicate that out of the three optimizations evaluated, using smaller quorums is the one that yields best performance enhancement. Using a rotating leader scheme does not seem to bring any benefit, and it is inconclusive whether or not removing communication steps improves performance. Finally, we showed that a standard BFT SMR protocol can display sufficiently predictable and acceptable latency in a WAN.

Acknowledgments

We warmly thank Marcos Vasco for his help in configuring PlanetLab's hosts and performing initial tests for us. This work was partially supported by the EC through projects TCLOUDS (FP7/2007-2013, ICT-257243), RC-Clouds (PCT/EIA-EIA/115211/2009) and also by FCT through the Multiannual Program (LaSIGE).

References

[1] BFT-SMART - High-performance Byzantine fault-tolerant State Machine Replication. http://code.google.com/p/bft-smart/.

[2] Emulab - Network Emulation Testbed. http://www.emulab.net/.

[3] D. Agrawal and A. F. Abadi. An efficient and fault-tolerant solution for distributed mutual exclusion. *ACM Transactions on Computer Systems*, 9(1):1–20, Feb. 1991.

[4] Y. Amir, C. Danilov, D. Dolev, J. Kirsch, J. Lane, C. Nita-Rotaru, J. Olsen, and D. Zage. Steward: Scaling byzantine fault-tolerant replication to wide area networks. *Dependable and Secure Computing, IEEE Transactions on*, 7(1):80–93, 2010.

[5] Y. Amir and A. Wool. Evaluating quorum systems over the internet. In *Proceedings of the The Twenty-Sixth Annual International Symposium on Fault-Tolerant Computing (FTCS '96)*, FTCS '96, pages 26–, Washington, DC, USA, 1996. IEEE Computer Society.

[6] O. Bakr and I. Keidar. Evaluating the running time of a communication round over the internet. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, PODC '02, pages 243–252, New York, NY, USA, 2002. ACM.

[7] O. Bakr and I. Keidar. On the performance of quorum replication on the internet. Technical Report UCB/EECS-2008-141, EECS Department, University of California, Berkeley, Oct 2008.

[8] C. Cachin. Yet another visit to Paxos. Technical Report Research Report RZ 3754, IBM Research Zurich, Nov. 2009.

[9] S. K. Card, G. G. Robertson, and J. D. Mackinlay. The information visualizer, an information workspace. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '91, pages 181–186, New York, NY, USA, 1991. ACM.

[10] M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions Computer Systems*, 20(4):398–461, Nov. 2002.

[11] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planetlab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, 33(3):3–12, July 2003.

[12] H. Garcia-Molina and D. Barbara. How to assign votes in a distributed system. *J. ACM*, 32(4):841–860, Oct. 1985.

[13] D. Gifford. Weighted voting for replicated data. In *Proc. of the 7th ACM Symposium on Operating Systems Principles*, pages 150–162, Dec. 1979.

[14] V. Hadzilacos and S. Toueg. A modular approach to fault-tolerant broadcasts and related problems. Technical report, Cornell University, May 1994.

[15] M. Herlihy. Dynamic quorum adjustment for partitioned data. *ACM Trans. Database Syst.*, 12(2):170–194, June 1987.

[16] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: Speculative Byzantine fault tolerance. In *Proc. of the 21st ACM Symp. on Operating Systems Principles - SOSP'07*, Oct. 2007.

[17] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[18] L. Lamport. The part-time parliament. *ACM Transactions Computer Systems*, 16(2):133–169, May 1998.

[19] L. Lamport. Paxos made simple. *ACM SIGACT News*, 32(4):18–25, Dec. 2001.

[20] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[21] Y. Mao, F. P. Junqueira, and K. Marzullo. Mencius: building efficient replicated state machines for wans. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, OSDI'08, pages 369–384, Berkeley, CA, USA, 2008. USENIX Association.

[22] R. B. Miller. Response time in man-computer conversational transactions. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, AFIPS '68 (Fall, part I), pages 267–277, New York, NY, USA, 1968. ACM.

[23] M. Naor and A. Wool. Access control and signatures via quorum secret sharing. *Parallel and Distributed Systems, IEEE Transactions on*, 9(9):909–922, 1998.

[24] J. Nielsen. *Usability Engineering*. Morgan Kaufmann Publishers, 1993.

[25] D. Peleg and A. Wool. Crumbling walls: a class of practical and efficient quorum systems. *Distributed Computing*, 10:87–97.

[26] J. Pâris. Voting with witnesses: A consistency scheme for replicated files. In *In Proceedings of the 6th International Conference on Distributed Computing Systems*, pages 3–6, 1986.

[27] G. Santos Veronese, M. Correia, A. Bessani, and L. C. Lung. EBAWA: Efficient Byzantine agreement for wide-area networks. In *High-Assurance Systems Engineering (HASE), 2010 IEEE 12th International Symposium on*, pages 10–19, 2010.

[28] F. B. Schneider. Implementing fault-tolerant service using the state machine aproach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, Dec. 1990.

[29] J. Sousa and A. Bessani. From Byzantine consensus to BFT state machine replication: A latency-optimal transformation. In *Proceedings of the Ninth European Dependable Computing Conference*, pages 37–48. IEEE, May 2012.

[30] G. Veronese, M. Correia, A. Bessani, L. C. Lung, and P. Verissimo. Efficient Byzantine fault-tolerance. *Computers, IEEE Transactions on*, 62(1):16–30, 2013.